

Dec 03, 01:022 ExternIterator.cpp Page 1/25

Dec 03, 01:022 ExternIterator.cpp Page 2/25

```
#include "ScriptRunner/ExternIterator.hpp"
#include "ScriptRunner/options.hpp"
#include "ScriptRunner/unpackTool.hpp"
#include "ScriptRunner/List.hpp"

#ifndef PHYSICSTOOL_H
#include "l3frootbase/physicsTool.hpp"
#endif
#ifndef L3PHYSICSRESULTS_
#include "l3results/L3PhysicsResults.hpp"
#endif
#include "l3run_config/L3RunConfigMgr.hpp"

// includes needed by the Parser
#include "l3parser/ParserPrimitives.hpp"
#include "l3parser/ParserStatic.hpp"
#include "l3registry/L3Factory.hpp"
#include "l3registry/dynload.hpp"
#include "l3filterbase/l3FilterMap.hpp"
#include "l3exceptions/L3ErrHandle.hpp"
#include <fstream>

// includes for streaming
#include "l3streaming/L3Streaming.hpp"
#ifndef ERROROBJ_H
#include "ErrorLogger/ErrorObj.hpp"
#endif
#ifndef ELSEVERITYLEVEL_H
#include "ErrorLogger/ELSeverityLevel.hpp"
#endif
#ifndef ZMEXERRORLOG_H
#include "ErLogEx/ZMExErrorLog.hpp"
#endif
#ifndef L3ZMXEL_H
#include "l3exceptions/L3ZMxel.hpp"
#endif

// some will be needed for chunks...
#include "l3chunk/L3Chunk.hpp"
#include "l3chunk/L3ChunkSelector.hpp"
#include "l3DebugChunk.hpp"
#include "l3chunk/L3MonChunk.hpp"
#include "ScriptRunner/L3MonData.hpp"

// those for L1L2bBlockData
#ifndef L1Crates
#include "edm/THandle.hpp"
#include "edm/RCPtr.hpp"
#include "q0om/d0_Ref.hpp"
#endif
#ifndef RawDataChunk
#include "l3base/RawDataChunk.hpp"
#endif
#ifndef RUN_CFG_MGR
#include "run_config_mngr/run_cfg_mgr.hpp"
#endif
#ifndef l1fw_util/l1fw_info.hpp"
#include "l1fw_util/l1fw_info.hpp"

```

Tuesday March 12, 2002

ExternIterator.cpp

1/13

Dec 03, 01:02:22

ExternIterator.cpp

Page 3/25

```

11info = new l1tfw_info;
12info = new l2fw_info;
event_header = new _ITC_EVENT_NAMESPACE::Event_Header;
itcunbiased = false;

// this set the default "detailed monitoring" & debuginfo frequency -
// can be overridden by SRDirective ( 0= never )
monitorInfo=10;
_debugInfo=1000;
_calib_on=false;
}

12itc = new unsigned int[l12base::MAX_NUM_L2_BITS];
13itc = new unsigned int[3*l2base::MAX_NUM_L2_BITS];
// initial pointers to external data to NULL
_runcfg = 0; // this does not appear to be used
_l2map = 0;
// Get instance of run_config from manager
int m = 0;
int c,s;
// createID = _runcfg->syscfg.ID/TRG_FR,0;
_createID = 31;
_useL1=false;
_useL2=false;
}

//-----ExternIterator::~ExternIterator()
{
    delete triglist;
    delete event_header;
    delete l1info;
    //delete l2info;
    delete [] l2itcc;
    delete [] l3itcc;
}

//-----ExternIterator::Set_Online(bool onl)
{
    online = onl;
    if(_online)
        _send_mon_info = true;
    else
        _send_mon_info = false;
}

//-----ExternIterator::operator++()
{
    _excurrent = _excurrent->_extlink;
    extIndex++;
    // Only valid (L2) bits should be considered
    while( _excurrent != 0 &&
          ! ( L2bits.test(bitsort[extIndex]) &&
               L2Eventbits.test(bitsort[extIndex]) ) )
    {
        _excurrent = _excurrent->_extlink;
        extIndex++;
    }
}

//-----ExternIterator::operator++() const
{
    linkNode *_ExternIterator::_Current() const
    {
        return _excurrent;
    }
}

//-----ExternIterator::operator++()
{
    _excurrent = _excurrent->_extlink;
    extIndex++;
    // Only valid (L2) bits should be considered
    while( _excurrent != 0 &&
          ! ( L2bits.test(bitsort[extIndex]) &&
               L2Eventbits.test(bitsort[extIndex]) ) )
    {
        _excurrent = _excurrent->_extlink;
        extIndex++;
    }
}

```

Dec 03, 01:02:22

ExternIterator.cpp

Page 4/25

```

//-----bool ExternIterator::isThere() {
//-----    return !(triglist->head ==0 );
//}

void ExternIterator::Begin() {
    if( (_excurrent = triglist->head ) == 0 )
    {
        ErrorObj maperror;
        maperror << endmsg;
        // throw it!
        ZMthrow( L3ZMxel ( maperror ) );
    }

    while( !_excurrent != 0 &&
          ! ( L2bits.test(bitsort[extIndex]) &&
               L2Eventbits.test(bitsort[extIndex]) ) )
    {
        _excurrent = _excurrent->_extlink;
        extIndex++;
    }
}

bool ExternIterator::isFinish() const {
    return !_excurrent != 0 ;
}

bool ExternIterator::Debug_On() {
    if( _calib_on )
        return true;
    if( _debugInfo == 0 )
        return false;
    if( _writeDebug || evcount%_debugInfo == 0 )
        return true;
    return false;
}

//-----bool ExternIterator::Traverse() {
//-----    //This is the main Internal Iterator method: it traverses the whole branch,
//-----    // sitting on each "real" Filter. executes DoThisFilter() and collect
//-----    // Comments: *local* convention: variables beginning with a "_" are pointers ex. -tail, -link
//-----    //ExtMap holds the Statistics for this L2 bit
//-----    LumMap *ExtLum =0;
//-----    LumMap *ExtLum_old =0;
//-----    L3StatMap *ExtMap =0;
//-----    typedef L3StatMap::iterator IIf;
//-----    clock_t beginTime;
//-----    bool first =true;
//-----}

```


Page 7/22

ExternalIterator.cpp

```

Dec 03, 01:02:22

// Asks if a special Filter is present,
if( strcmp(_tmp->FilterType(), "L3Unbiased") == 0 ) {
    L3Unbiased = true;
    _writeDebug = true;
    // true for the whole branch
    if( ResultSoFar) {
        Unbiased = true;
        //this Filter must pass always
        ResultSoFar = true;
    }
}

else if( strcmp(_tmp->FilterType(), "prescale") == 0 ) {
    if( !ResultSoFar) {
        Prescaled = true;
        // prescale flag precede all others
        GoToMother = true;
    }
}

// end if special

Unbiased = Unbiased || L2Unbiased ;
nodeStat->passed = ResultSoFar;
isScriptPassed = isScriptPassed && ResultSoFar;

// If this Filter FAILS, dont proceed from here, if not Unbiased
if( !ResultSoFar && !Unbiased ) {
    GoToMother = true;
}

#ifdef L3DEBUG
    std::cout << _tmp->ID << " failed..." << std::endl;
#endif

if( !_prescaled) {
    if( strcmp(_tmp->FilterType(), "fake") != 0 ) {
        if( strcmp(_tmp->FilterType(), "filter") != 0 ) {
            //filters will know later about their status
            isScriptUnbiased = isScriptUnbiased && Unbiased;
            if( ResultSoFar )
                nodeStat->NbPassed++;
        }
    }
}

//if didn't pass but was unbiased, then mark it as force_unbiased
if( Unbiased ) {
    if( L2Unbiased)
        nodeStat->L2Unbiased = true;
    if( L3Unbiased)
        nodeStat->L3Unbiased = true;

    if( !ResultSoFar ) {
        strcpy(BitStatus, "force_unbiased");
    }
    else {
        //First, ask if this branch was marked force_unbiased already
        if( strcmp(BitStatus, "force_unbiased") !=0 )
            strcpy(BitStatus, _tmp->FilterType());
    }
}

// end !fake
if( !_prescaled )
}

// end !first
}

```

----- End node processing -----

```

Dec 03, 01 0:22                                         ExternIterator.cpp   Page 8/25
} else { // ==0 or GotoMother
    // end of this branch or this filter didn't pass
    _current = _current->_mother;
    last = true;
}
// Here we are back to the Filter script,
// let's collect statistics for it *now*
NodeStat *nodeStat = &(*ExtMap->find(_current->_GetPt()->ID)) .second;

LumStat *lumiStat;
if( _old_lum )
    lumiStat = &(*ExtLum_old->find(_current->_GetPt()->ID)) .second;
else
    lumiStat = &(*ExtLum->find(_current->_GetPt()->ID)) .second;

lumiStat->NbCalled++;
strcpy(nodeStat->status, BitStatus);

if( Prescaled )
    nodeStat->NbPreScale++;

if( strcmp(BitStatus, "force_unbiased") == 0 ) {
    // write out this event
    _passed = true;
    nodeStat->Nb3Unbiased++;
}

if( GotoMother ) {
    //FAIL
    nodeStat->passed = false;
    lumiStat->NbFailed++;
}

else {
    //here if passed or unbiased
    // that is, the iterator reached the end of the branch
    _passed = true;
    nodeStat->NbPassed++;
    lumiStat->NbPassed++;
}

nodeStat->passed = isScriptPassed;
nodeStat->L3prescale = Prescaled;
if( isScriptUnbiased ) {
    if(L2unbiased)
        nodeStat->L2Unbiased = true;
    if(L3unbiased)
        nodeStat->L3Unbiased = true;
}
if( isScriptUnbiased )
    ITCunbiased = true;

#ifndef L3FDEBUG
std::cout << "Going back..." << std::endl;
#endif
} //end else
-----}
else { //if last
    // last node of this branch
    current = _start;
}
// Done
if( start->branch == 0 ) {

```

Dec 03, 01:02:22 ExternIterator.cpp Page 9/25

```

} else {
    // if ANOTHER branch
#endif L3FDEBUG std::cout << "OOPS, another branch" << std::endl;
#endif // If ONE Filter script pass, that's enough,
// keep this event
if( ResultSoFar )
    _passed = true;

// points to start to proceed, anyway
_current = _start->_branch ;
last = GoToMother = Unbiased = L3Unbiased = Prescaled = false;
isScriptPassed = isScriptUnbiased = true;
strcpy(BitStatus, "0");
// resume with this new branch
_start = _current;

#endif L3FDEBUG std::cout << "End this traverse..." << std::endl;
#endif
// end traversing

} // end if last
// -----
// -----
// ----- end while
_passed = _passed || ResultSoFar;
return true;
}

// -----
// -----
// ----- ExternIterator::L3ChunkGen( edm::Event *event ) {
// those go to Chunk
L3Chunk::L3ChunkPhysToolMap _physToolMap;
L3Chunk::L3ChunkFilterMap _map;

//~/l3streaming
L3Streaming *l3streaming = L3Streaming::instance();
LogicalStream logicalStream;
typedef l3map<const l3string, bNode *> const_iterator CIFT;
int _physcand;
bool _ok = false;

// get the Event location from the unpacker
// edm::Event *event = unpackTool->evLoc;

//Tools section

dynlk::ToolFactory *fact = dynlk::ToolFactory::instance();
for ( dynlk::ToolFactory::tool_const_iterator pt= fact->tool_begin(); pt!= fact->tool_end(); ++pt ) {

```

```

Dec 03, 01:02:22                                         ExternIterator.cpp
Page 10/25

bTool *ttc = (*pt).second;

if ( ttc->HaveIRun() ) {
    if ( ttc->GetType() == "physics" ) {
        // here an electron type Tool
        std::list<L3PhysicsResults*> results;
        // no run-time typechecking (which would fail)
        physicsTool<L3PhysicsResults> * test =
            static_cast<physicsTool<L3PhysicsResults>> ( ttc );
        if ( test != 0 ) {
            physcand = test->get_toolResults()->get_n_candidates();
            if ( _physcand >0 ) {
                // physcand >0
                // test method
                -ok = true;
                test->WritePhysics(results);
                _physToolMap[ttc->GetId()] = results;
            }
            // ...
        } else {
            _IteratorLog(ELerror, "Couldn't cast tool to a physictool!")
            << endlmsg;
        }
    }
}

//Filter section
L3StatMap::const_iterator p;

int ii;
for( ii=0; ii< extIndexMax ; ii++ ) {
    std::map<std::string,NodeStat> tempo;
    int this12bit=bitsort[i];
    for( p = SumStat[this12bit].begin() ; p!=SumStat[this12bit].end() ; ++p
) {
    L3String _s_first= (*p).first;
    tempo[_s_first]=(*p).second;
    //~L3streaming
    if( _s_first == (*p).second.mother //a l3 trigger
&& ( *p ).second.passed ) { //and passed
        logicalStream |=
            L3streaming->getScriptStreamType(_s_first);

    //L3_ITC info
    13itc[13count]=_l3map[_s_first];
    13count++;
}
}
}

-----
// Make a chunk and insert it
std::auto_ptr<L3Chunk> 13stuff( new L3Chunk() );
if ( _ok ) //at least one physcand>0
    13stuff->insertPhyTool(_physToolMap);
}
}

```

Dec 03, 01:022 ExternIterator.cpp Page 11/25

```

13stuff->insertFilterData( _map );
}

//~13streaming
13stuff->insertLogicalStream( logicalStream );
13streaming->record(logicalStream);

//Label the Chunk with "online" or "offline"
if( _online )
13stuff->insertLabel("online");
else
13stuff->insertLabel("offline");

edm::ChunkID id1 = edm::insertChunk(*event, 13stuff);

#ifdef DEBUG_OUT
// Get back L3Chunk and check.
L3ChunkSel s1_off("offline");
L3ChunkSel s1_on("online");

edm::TKey<L3Chunk> l3key_off(s1_off);
std::list<edm::THandle<L3Chunk>> chlist_off;
l3key_off.findall(*event, chlist_off);

-IteratorLog(ELinfo, "Found") << chlist_off.size() << "'offline' L3Chunks in the even
t" << endlmsg;
std::list<edm::THandle<L3Chunk>> chlist_on;
l3key_on.findall(*event, chlist_on);

-IteratorLog(ELinfo, "Found") << chlist_on.size() << "'online' L3Chunks in the event"
<< endlmsg;
#endif
return true;
}

```

```

-----
-----
bool ExternIterator::L3DebugChunkGen( edm::Event *event ) {

```

```

    bool ok = false;
    L3DebugInfo * its_info;

```

```

    //Make a Chunk
    std::auto_ptr<L3DebugChunk> l3stuff( new L3DebugChunk() );

```

```

    //Loop over Tools
    dynlk::ToolFactory *fact = dynlk::ToolFactory::instance();

```

```

    for( dynlk::ToolFactory::const_iter p= fact->tool_begin(); p!= fact->tool_end(); p++ ) {

```

```

        13string tkey = p->second->GetId();
        p->second->Write( &its_info );
    }

```

```

    //only deal with no null debug area
    if( its_info != 0 ) {

```

```

        if( p->second->HaveIRun() && its_info->isComplete() ) {

```

```

            // insert it
            13stuff->insertDebugInfo( tkey, its_info );
            ok = true;
        }
    }

```

```

#ifdef L3DEBUG
else {
    std::cout << p->second->GetId() << " Debug Chunk : Not complete " << std
    ::endl;
#endif

```

Dec 03, 01:022 ExternIterator.cpp Page 12/25

```

    } //end of !0
} //end for

if(ok) {
    edm::ChunkID id1 = edm::insertChunk(*event, 13stuff) << endlmsg;

#ifdef DEBUG_INFO
//Test: Get back L3DebugChunk and check.
edm::TKey<L3DebugChunk> l3key;
edm::THandle<L3DebugChunk> l3chk = 13Key.find( *event );
if( l3chk.isValid() ) {
    std::cout << "\nL3DebugChunk printChunk, after insertion and retrieval: " << endl;
    l3chk->printChunk( cout );
}

L3DebugChunk &chttest = const_cast<L3DebugChunk &>(*l3chk);
L3DebugChunk::L3DebugChunkMap *mtest = chtest.getDebugInfo();

for(L3DebugChunk::map_iter pp = mtest->begin(); pp != mtest->end(); pp++) {
    if( pp->first == "SMTHpData" ) {
        L3SmtUpnPDebugInfo *detest = dynamic_cast<L3SmtUpnPDebugInfo * >
        (pp->second);

        if( detest != 0 )
            detest->printSummary( cout );
    }
    else
        std::cout << "IRKK!!" << endl;
}
return true;
}
else {
    -IteratorLog(ELinfo, "In L3DebugChunk: L3CDDebughunk not found in Event" )
    << endlmsg;
    return false;
}
#endif
-----
void ExternIterator::ITOL2Bits()
{
    -----
    bool ExternIterator::genEventHeader( const edm::Event&edmevent ) {
        // first fill l3 bit "255" (as requested by M.Begin)
        if(IRunbiased)
            13itc[13count] = 255;
        13count++;
    }
}

```

Dec 03, 01:22

ExternIterator.cpp

Page 13/25

```
// fill in the 11/12/13 bits info:
event_header->N_L1(12count,12itc); //MUST MODIFY !!! ( good for now )
event_header->N_L2(12count,12itc);
event_header->event(L1EventNb);
event_header->lum_block(LumIndex);
event_header->N_L3(13count,13itc);

event_header->run(0);
event_header->data_size(0);

//here is the Zhong's streaming:
//get 13 chunk

edm::THandle<L3Chunk> h_l3chunk;
edm::TKey<L3Chunk> edmkey(edmevent);

#ifndef L3F_OFFLINE
h_l3chunk = edmkey.find(edmevent);
#else
L3Chunksel sl_off("offline");
edmkey:TKey<L3Chunk> l3key(sl_off);
std::list<edmkey:THandle<L3Chunk>> chlist_off;
l3key_off.findAll(edmevent, chlist_off);
if(chlist_off.size() > 0)
    h_l3chunk = *chlist_off.begin();
else
    h_l3chunk = edmkey.find(edmevent);
#endif

//const cast
L3Chunk &l3chunk = const_cast<L3Chunk&>(*h_l3chunk);

//stream
L3Streaming *l3streaming = L3Streaming::instance();
l3streaming->physical->DS
13string stream_name =
l3streaming->logical12physical(13chunk.getLogcalStream()).getName();
const l3vector<UINT> *pStrmID =
l3streaming->getStreamIDs(stream_name);

if(pStrmID == 0){
    ErrorObj maperror( EAbort,
        "Error - Failing to get Stream IDs" );
    maperror << endlmsg;
    event_header->run(77);
    return false;
} else {
    Nstrm = pStrmID->size();
    UINT *strm = new UINT[Nstrm];
    for(int i=0; i<Nstrm; i++)
        strm[i] = (*pStrmID)[i];
    event_header->N_streams(Nstrm,strm);

    // release memory
    delete strm;
}
```

Dec 03, 01:022

ExternIterator.cpp

Page 14/25

```
// fill in the 11/12/13 bits info:
    return true;
}
//-----
void ExternIterator::ResetMyself() {
    typedef L3StatMap::iterator II;

    int ii;
    // we want just to reset last (extIndexMax) active 12bits, so we
    // have to use bitsort
    for( ii=0; ii< extIndexMax ; ii++ ) {
        for( II p= SumStat[bitsort[ii]].begin(); p!=SumStat[bitsort[ii]].end() ; ++p ) {
            (*p).second.mother="";
            strcpy((*p).second.status, "(0)");
            (*p).second.passed = false;
            (*p).second.called = false;
            (*p).second.L3Unbiased = false;
            (*p).second.L2Unbiased = false;
        }
    }
    //-----
    passed = false;
    writeDebug = false;
    L2Unbiased = false;
    extIndex = 0;
    l2count = 0;
    l3count = 0;
    // L2Eventbits.reset();
}

//-----
void ExternIterator::ResetTools() {
    dynlk::ToolFactory *fact = dynlk::ToolFactory::instance();
    for( dynlk::ToolFactory::tool_const_iter
        pt= fact->tool_begin(); pt!= fact->tool_end(); ++pt ) {
        bTool *tmp = (*pt).second;
        tmp->written =false;
        tmp->ToolReset();
    }
}

//-----
void ExternIterator::ToolsRunInit() {
    dynlk::ToolFactory *fact = dynlk::ToolFactory::instance();
    for( dynlk::ToolFactory::tool_const_iter
        pt= fact->tool_begin(); pt!= fact->tool_end(); ++pt ) {
        13string _toolType = typeid(*pt).name();
        // some platforms return "class <type_name>" ;
        if( __toolType.compare(0,5,"class") == 0 )
            __toolType = __toolType.substr(6,__toolType.size()-5);

        bTool *tmp = (*pt).second;
        if(_toolType != "L3RunConfigMgr" ) // don't do L3RunConfigMgr RunInit()
            tmp->RunInit();
    }
}
```

Dec 03, 01:02:22 ExternIterator.cpp Page 15/25

Dec 03, 01:02:22 ExternIterator.cpp Page 15/25

```

void ExternIterator::ResetFilters() const {
    typeDef<13map<const L3String, bNode *>>::const_iterator CI ;
    for( CI p=13ffFilterMap::register_.begin(); p!=13ffFilterMap::register_.end(); ++p ) {
        (*p).second->Reset();
    }
}

void ExternIterator::Destroy() {
    std::bitset<12base::MAX_NUM_L2_BITS> drop_all;
    drop_all.set();
    DropL2MonInfo( drop_all );
    DropL2Bits( drop_all );
}

// Drop any Tools left, if any
#ifndef _DEBUG
__IteratorLog(ElInfo, "Begin final cleanup, from here") << endl;
#endif

DynLoad dl;
ToolFactory *fact = dynlk::ToolFactory::instance();
// Force all Tools to have reference count = 0 ( == 1 )
for(std::map<const std::string, int>::iterator p = fact->_tREF.end(); p++ ) {
    std::cout << "Survivor.reference count = " << p->first << " - " << p->second << endl;
}
p->second = 1;

for( int ii=0; ii< extIndexMax ; ii++ )
    SumStat[bitsort[ii]].clear();

}

// -----
void ExternIterator::Summary() {
    L3MonChunk chnk1;
    getLumData(chnk1);
    chnk1.printchunk(std::cout);
}

// -----
bool ExternIterator::BeginRun( std::bitset<12base::MAX_NUM_L2_BITS> & thisbit ) {
    // reference count it
    for( int ii=0; ii<12base::MAX_NUM_L2_BITS; ii++ ) {
        if( thisbit.test(ii) )
            _l2ref[ii]++;
        L2bits = L2bits | thisbit;
    }
}

// -----
bool ExternIterator::EndRun( std::bitset<12base::MAX_NUM_L2_BITS> & thisbit ) {
    //only deactivate l2bits with ref_count=0
    for( int ii=0; ii<12base::MAX_NUM_L2_BITS; ii++ ) {
        if( thisbit.test(ii) ) {
            _l2ref[ii]--;
            if( _l2ref[ii] == 0 ) {
                L2bits.reset(ii);
            } else if( _l2ref[ii]<0 ) {
                _IteratorLog(Fatal, "StopRun: Attempt to deactivate no active L2 bit %d" << ii << endl);
                return false;
            }
        }
    }
}

// -----
bool ExternIterator::EndRun( std::bitset<12base::MAX_NUM_L2_BITS> & thisbit ) {
    //only deactivate l2bits with ref_count=0
    for( int ii=0; ii<12base::MAX_NUM_L2_BITS; ii++ ) {
        if( thisbit.test(ii) ) {
            _l2ref[ii]--;
            if( _l2ref[ii] == 0 ) {
                L2bits.reset(ii);
            } else if( _l2ref[ii]<0 ) {
                _IteratorLog(Fatal, "StopRun: Attempt to deactivate no active L2 bit %d" << ii << endl);
                return false;
            }
        }
    }
}

// -----
void ExternIterator::UpdateL2Map(13map<int, L3String> &_l2tmp ) {
    _l2tmp = &_l2tmp;
}

// -----
void ExternIterator::UpdateL3Map( 13map<int, ClientDef *> &tmp ) {
    _l3map.clear();
    for( 13map<int, ClientDef *>::const_iterator it = tmp.begin(); it != tmp.end(); ++it ) {
        13map<int,L3String> *mp = &it->second->_l3map;
        for( 13map<int,L3String>::const_iterator itt = mp->begin(); itt != mp->end(); ++itt ) {
            _l3map[itt->second] = itt->first;
        }
    }
}

// -----
bool ExternIterator::getLumData() {
    if( ! _send_mon_info )
        return false;
    if( _new_lum ) {
        _new_lum = false;
        return true;
    } else
        return false;
}

// -----
bool ExternIterator::getLumData( L3MonChunk & chnk ) {
    L3MonChunk::L3ChunkStatMap _map;
    std::map<const char *, NodeStat>::const_iterator p ;
    int ii;
    lumcount++;
    for( ii=0; ii< extIndexMax ; ii++ )
        L3MonChunk::StratMap tempo;
}

```

```

Dec 03, 01:02:22          ExternIterator.cpp      Page 17/25
int this12bit=bitsort[i];
for( p = SumStat[this12bit].begin() ; p!=SumStat[this12bit].end() ; ++p
){
    tempo[(*p).first]=(*p).second;
    _map[this12bit]=tempo;
}
if (_12map==0)
{
    ErrorObj.l2maperror( "Elsewhere",
    "ExternIterator::getLumData: Error _l2map has not been initialised" );
    l2maperror <<
    " " : probable failure in triggerlist parsing" << endlmsg;
    ZMthrcow( L3ZNxel( l2maperror ) );
}
chnk.insertL2Map(*_12map);
chnk.insertStatData(_map);
chnk.insertLum(_lumStat_old, _lumStat_old_save);
// if (_lumStat_old.size() != 0 )
// std::cout << "Test : "
// (*_*lumStat_old.begin()).second.begin() .second.begin() .second.begin()
dl;
if (_monitorInfo == 0 )
    return true;
if( lumcount%monitorInfo ==0 )
{
    monId=1;
    //here for "complete Monitor"
    13StatManager *_statmanager = 13StatManager::Instance();
    chnk.insertToolData(_statmanager->timeclassmap);
    chnk.insertFilterData(_statmanager->filterdataclassmap);
}
return true;

```

```

//-----ExternIterator::setMonMap()
void ExternIterator::setMonMap() {
}

void ExternIterator::setMonMap( int _rnb, const std::bitset<12base> :MAX_NUM_L2_BI
    TS> & _set) {
    _12run_map[ _rnb ] = _set;
}

//----- Parsing trigger list - called by the ScriptRunner constructor
// Carmen Silva 05/05/98
bool ExternIterator::AddL2bits(L3string &TriggerList) {

    L3string bitmap = "";
    DynLoad dl;
    L3Streaming *l3streaming = L3Streaming::instance(); //~l3streaming

    ParserPrimitives pl;
    ParserStatic *parserNgr = ParserStatic::instance();
    if( ! parserNgr->openTrigList(TriggerList) ) {
        ErrorObj maperror( Elfatal, "Can't Open TrigList File ! " );
        maperror << TriggerList << endlmsg;
        // throw it !
        //ZMthrow( LZMxel (maperror));
        return false;
    }
}

```

```

Dec 03, 01 0:22          ExternIterator.cpp      Page 18/25
} l3string ch;

bool _endfile = false;
_endfile = parserMgr->get_valid_line ( ch );

while ( !_endfile ) {
    if ( pl.is_directive(parserMgr->par_value[1]) ) {
        // First, check if it is just a SR directive -----
        int ii = 1;
        while( parserMgr->par_value[+ii] != " " ) {
            if( parserMgr->par_value[ii] == "monitorinfo" ) {
                _monitorInfo=atoi(parserMgr->par_value[ii+1].c_str());
            } else if( parserMgr->par_value[ii] == "sendmoninfo" ) {
                if(parserMgr->par_value[ii+1]== "yes" )
                    _send_mon_info=true;
                else
                    _send_mon_info=false;
            } else if( parserMgr->par_value[ii] == "setonline" ) {
                if(parserMgr->par_value[ii+1]== "yes" )
                    _online=true;
                else
                    _online=false;
            } else if( parserMgr->par_value[ii] == "calib" ) {
                if(parserMgr->par_value[ii+1]== "yes" )
                    _calib_on=true;
                else
                    _calib_on=false;
            } else if( parserMgr->par_value[ii] == "useL1" ) {
                if(parserMgr->par_value[ii+1]== "yes" )
                    _useL1=true;
                else
                    _useL1=false;
            } else if( parserMgr->par_value[ii] == "useL2" ) {
                if(parserMgr->par_value[ii+1]== "yes" )
                    _useL2=true;
                else
                    _useL2=false;
            } else if( parserMgr->par_value[ii] == "debuginfo" ) {
                _debugInfo= atoi(parserMgr->par_value[ii+1].c_str());
            } else {
                _IteratorLog(ELwarning, "Invalid SRDirective "
                            << parserMgr->par_value[ii] << " -> IGNORED" << endl
                            << endl);
            }
        } -----
        //----- Here a L2 trigger bit:
        #ifndef P_DEBUG
        std::cout << "Instantiating L2 bit"
        #endif
    } -----
}

```

Dec 03, 01:022 ExternIterator.cpp Page 19/25

Dec 03, 01:022 ExternIterator.cpp Page 20/25

```

// Here we test for another branch of the same trig_bit
if( parserMgr->trigbit == bittmp )
    triglist->Append( " " );
else {
    triglist->Append(parserMgr->trigbit);
    // build L2 bit sorting map
    l3string L2strg = parserMgr->trigbit;
    if( _l2map==0 ) {
        ErrorObj l2maperror( ESevere, "ExternIterator::AddL2Bits : Error _l2map has not been initialised" );
        l2maperror << " " << L2strg << " : probable failure in triglist parsing" <
        < endmsg;
        ZMthrow( L3ZNxel( 12hmaperror ) );
    }
    for( l3map<int, l3string>::const_iterator it_map
        = _l1map->begin(); it_map != _l2map->end(); it_map++ ) {
        if( it_map->second == L2strg ) {
            bitsort[extIndexMax]= it_map->first;
        }
    }
    if(bitsort[extIndexMax] <0 ) {
        ErrorObj maperror( ESevere, "FATAL - SOMETHING WRONG : " );
        maperror << L2strg << " Probably a configuration file <-> triglist mismatch" <
        < endmsg;
        // throw it!
        //ZMthrow( L3ZNxel ( maperror ) );
        return false;
    }
    extIndexMax++;
}
bittmp = parserMgr->trigbit;
}

// Here necessarily follows a L3 primary Filter declaration
parserMgr->reset_arrays();
_endfile = parserMgr->get_valid_line( ch );
l3string primary_filter;
if( parserMgr->is_valid_primary_filter(primary_filter) ) {
    #ifdef P_DEBUG
    std::cout << "Instantiating primary filter"
    << primary_filter << std::endl;
    #endif
    triglist->AppendMother( primary_filter );
    l3streaming->addTrigger( primary_filter ); //~l3streaming
} else {
    ErrorObj maperror( ESevere,
        "AddL2Bits ->Should have 'filter' after trigger bit " );
    maperror << parserMgr->trigbit << endmsg;
    // throw it!
    //ZMthrow( L3ZNxel ( maperror ) );
    return false;
}
} else if(parserMgr->get_valid_filter_refset() ) {
    #ifdef P_DEBUG
    parserMgr->filter_name=parserMgr->par_value[1];
    #endif
}

// Here Filters get instantiated
#endif
// Here Tools get instantiated
#endif
// Here Tools get instantiated
#endif
// L3RunConfig initialization
if( ttype == "L3RunConfig" * _refL3RCM = dynamic_cast<L3RunConfigMgr * >
    ( dl.addToolInstance( parserMgr->par_value[1], parserMgr->tool_name ) );
    return false ; // something 'severely' wrong - see log_file
}

if( _refL3RCM == 0 ) {
    ErrorObj maperror( Elabot, "AddL2Bits -> NO L3RunConfigMgr registered!!" );
}

// throw it!
//ZMthrow( L3ZNxel ( maperror ) );
return false;
}

_errorObj _refL3RCM->RunInit( _createList );
}

} else {
    ErrorObj maperror( ESevere,
        "Trigger List No Such Tool or Filter: " );
    maperror << parserMgr->trigbit << endmsg;
    // throw it!
    //ZMthrow( L3ZNxel ( maperror ) );
    return false;
}

// parserMgr->reset_arrays();
_endfile = parserMgr->get_valid_line( ch );
}

parserMgr->closeTrigList();
return true;
}

// begin L2 traverse
_linkNode * _trig;
_excurrent = _triglist->head ;
int ii =0;

bool _dropLeft = false;
_linkNode * _trig;
}

void ExternIterator::DropL2Bits( std::bitset<12base>::MAX_NUM_L2_BITS> &Dropit )
{
}

```

ExternIterator.cpp

```

linkNode * _previous = _excurrent;

while( !_excurrent != 0 ) {
    if( Dropit.test(bitsort[iii]) ) {  

        #endif  

        // starting point for this L2  

        _current = _start = _excurrent  

        bNode * _tmp = _current->_GetPnt  

        bool first = true;  

        bool last = false;  

        bool done = false;  

        bool GoToMother = false;  

  

        while( ! done ) {  

            if( !last ) {  

                if( _current->_link != 0 &  

                    ! GotoMother ) {  

                    // -----Regular Node-----  

                    if( first ) {  

                        // Trig node - point to  

                        // Regular node -  

                        trig = _current = _s  

                        First = false;  

                    } else {  

                        // Regular node - poi  

                        linkNode * _killed = _  

                        _current = _current->  

                        // don't kill trigs at  

                        if( !_killed != _killed  

                            l3ffFilterMap:Drop  

                        delete _killed;  

                    }  

                    // here the pointer to  

                    _tmp = _current->_Get  

#ifdef L3FDEBUG  

                    _tmp->hello();  

#endif  

                } // end !first  

            } // -----End node-----  

        } // end of this branch  

        linkNode * _killed = _cu  

        _current = _current->_mo  

        // in case of an empty (
```

ExternIterator.cpp

```

    13FFilterMap::Drop( _tmp );
    delete _killed;
}

// delete this filter script
13FFilterMap::Drop(_current->_GetPT());
delete _current;
last = true;

#ifdef L3FDEBUG
    std::cout << "Going back..." << std::endl;
#endif

// -----
} else {
    // last node of this branch
    current = start;

    // Done
    if( _start->_branch == 0 ) {
        done = true;
    } else {
        // if ANOTHER branch
        #ifdef L3FDEBUG
            std::cout << "OOPS, another branch" << std::endl;
        #endif
        // points to start to proceed, anyway
        current = _start->_branch ;
        last = GoToMother = false;
        // resume with this new branch
        _start = current;
    }
    #ifdef L3FDEBUG
        std::cout << "End this traverse..." << std::endl;
    #endif
    // end traversing
}
} // end if last
// -----
} // end L3 while

// erase iith element from bitsort
for (int jj=ii; jj < l2base;MAX_NUM_L2_BITS-1 ; jj++)
    bitsort[jj] = bitsort[jj+1];
// re-initialize last element
bitsort[l2base;MAX_NUM_L2_BITS-1] = -1;
extIndexMax--;
ii--;

// update triglist "head", if it's the case
if( _excurrent == triglist->_head)
    triglist->_head = _excurrent->_extlink;
// end Dropit
#ifdef L3FDEBUG
    //if not dropped, increment previous
#endif
}

```

```

Dec 03, 01:02:22 Page 23/25
ExternIterator.cpp

void ExternIterator::Dropl2MonInfo(std::bitset<L2base::MAX_NUM_L2_BITS> & DropInfo)
{
    int previous = _extcurrent;
    int i;
    for( i=0; i<L2base::MAX_NUM_L2_BITS ; i++ ) {
        if(Dropit.test(i)) {
            //ifdef _DEL_DEBUG
            //    _IteratorLog(ELinfo,"Dropping L2 bit Monitoring Info ")
            //endif
            //    << _l2map->find(i)->second << endlmsg;
            SumStat.erase(i);
        }
    }
    //ExternIterator::getL2Info(const editm::Event &event) {
    if( _useLL ) {
        _old_lum = false;
        unsigned int lumtmp = LumIndex;
        if( ! liinfo->ForEventHeader( event, L2Eventbits,
                                       L1EventNb, LumIndex ) ) {
            _IteratorLog(ELwarning , "l1info returned false!!")
            << " L1 bit mask is being reset to 0" << endlmsg;
        L2Eventbits.reset();
        L1EventNb=0;
        LumIndex=0;
        return false;
    } else {
        #ifdef L3FDEBUG
        std::cout << "event Number" << L1EventNb << " - LumIndex " << LumIndex << std::endl;
        #endif
        if( LumIndex != _l1bits ) {
            new_lum = true;
            if( LumIndex == _oldlum ) {
                _old_lum = true;
            } else {
                _oldlum = _lumtmp;
                //put lmap into old_map and reset
                _lumStat_old_save = _lumStat_old;
                _lumStat_old = _lumStat;
            }
        }
        int ii;
        for( ii=0; ii< extIndexMax ; ii++ ) {
            .begin();
            it!=_lumStat[bitsort[ii]].end(); it++ ) {

```

```

Dec 03, 01 0:22          ExternIterator.cpp      Page 25/25

        }

    }  

    if( L2Eventbits.test(127) )  

        L2Unbiased = true;  

  

} else {  

    // no information, set all L2 bits  

    L2Eventbits.set();  

    L1EventNb=0;  

    LumIndex=0;  

}  

  

return true;  

}

void ExternIterator::setupTest() {
    L3Streaming *l3streaming = L3Streaming::instance(); //~l3streaming
    if(_l2map==0)
    {
        ErrorObj l2maperror( Elsevere, "ExternIterator::setupTest : Error _l2map has not been
initialised" );
        l2maperror << " : probable failure in triggerlist parsing" << endlmsg;
        ZMthrow( L3ZnXel(12maperror) );
    }
    for( l3map_int i3string::const_iterator p=
        _l2map->begin(); p!=_l2map->end(); p++ )
    {
        triglist->Append(p->second);
        // build L2 bit sorting map
        bitsortIndexMax=p->first;
        extIndexMax++;
    }
    triglist->AppendMother( "daq_test" );
    l3streaming->addTrigger( "daq_test" ); //~l3streaming
    triglist->AppendSister( "MarkPass_all", "L3FMarkAndPass" );
    // l3streaming->addFilter();
    //?????
}
}

//-----
int ExternIterator::getMonId()
{
    int tmp=monid;
    monid = 0;
    return tmp;
}

```